USC Viterbi
School of Engineering

INF558 PAPER PRESENTATION

SCALABLE SEMANTIC WEB DATA MANAGEMENT USING VERTICAL PARTITIONING

KARAN MAHESHWARI

04/09/2019

Topics covered:

1) RDF Data

2) Performance and scalability Issues while storing RDF Data

3) Solutions

RDF Data

Format: *<subject, property, object>*

Advantages:

- Easy to convert any kind of data (relational, XML, etc.) to this format of triples
- Easy to build tools that manipulate RDF Formatted data

RDF Data

Disadvantages:

- Simplest queries require multiple joins

Example: Find Books by Serge that have "Foundation" in title.

person1	isNamed	"Serge Abiteboul"
person2	isNamed	"Rick Hull"
book1	isType	"Book"
book2	isType	"Book"
book3	isType	"Book"
book1	hasAuthor	person1
book3	hasAuthor	person1
book1	isTitled	"Foundations of Databases"
book2	isTitled	"Advanced Database Systems"

As the size and number of "property" increases, more and more join are required. Complex queries will require even more joins.

Solutions?

1) Property Tables
2) Vertically partitioning the RDF data

1) Property Tables

Methodology: Widen the triples table into a conventional traditional DBMS format with 1 column for every "property"

Example:

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

Subject	Type	Title	Copyrt	Author	Artist	Lang.
ID1	Book	"XYZ"	"2001"	"Fox, Joe"	NULL	NULL
ID2	CD	"ABC"	"1985"	NULL	"Orr, Tim"	"French"
ID3	Book	"MNP"	NULL	NULL	NULL	"English"
ID4	DVD	"DEF"	NULL	NULL	NULL	NULL
ID5	CD	"GHI"	"1995"	NULL	NULL	NULL
ID6	Book	NULL	"2004"	NULL	NULL	NULL

Problems:

- Too Many Nulls.

  Can be Reduced by making columns for only frequently queried "property" or or only those "property" that are present in most of the subjects.

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Property Table

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID2	language	"French"
ID3	language	"English"

Left-Over Triples

- Multivalued attributes are still problematics since multiple rows for them.

Another form of Property Table called Class Property Table

Class: BookType

Subj.	Title	Author	copyright
ID1	"XYZ"	"Fox, Joe"	"2001"
ID3	"MNP"	NULL	NULL
ID6	NULL	NULL	"2004"

Class: CDType

Subj.	Title	Artist	copyright
ID2	"ABC"	"Orr, Tim"	"1985"
ID5	"GHI"	NULL	"1995"

Left-Over Triples

Subj.	Prop.	Obj.
ID2	language	"French"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"

2) A) Vertically Partitioned Approach

Methodology: The triples table is rewritten into *n* two column tables where *n* is the number of unique properties in the data.

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

Type

ID1	BookType
ID2	CDType
ID3	BookType
ID4	DVDType
ID5	CDType
ID6	BookType

Author

ID1	"Fox, Joe"

Title

ID1	"XYZ"
ID2	"ABC"
ID3	"MNO"
ID4	"DEF"
ID5	"GHI"

Artist

ID2	"Orr, Tim"

Copyright

ID1	"2001"
ID2	"1985"
ID5	"1995"
ID6	"2004"

Language

ID2	"French"
ID3	"English"

Each table is sorted by subject, so that particular subjects can be located quickly, and that fast merge joins can be used to reconstruct information about multiple properties for subsets of subjects.

Advantages:

- Support for multi-values attributes.
- No need to store NULL's.
- No clustering algorithms are needed.

Disadvantages:

- Although merge-join can be done easily, it is still an expensive operation.
- Inserts can be slower as multiple tables have to be updated. (Although not really a problem as a data usually doesn't come at such a high rate)

2) B) Vertically Partitioned Approach by Extending a column-oriented DBMS

Methodology: Use column-oriented databases to store tables as collections of columns rather than as collections of rows.

Advantages:

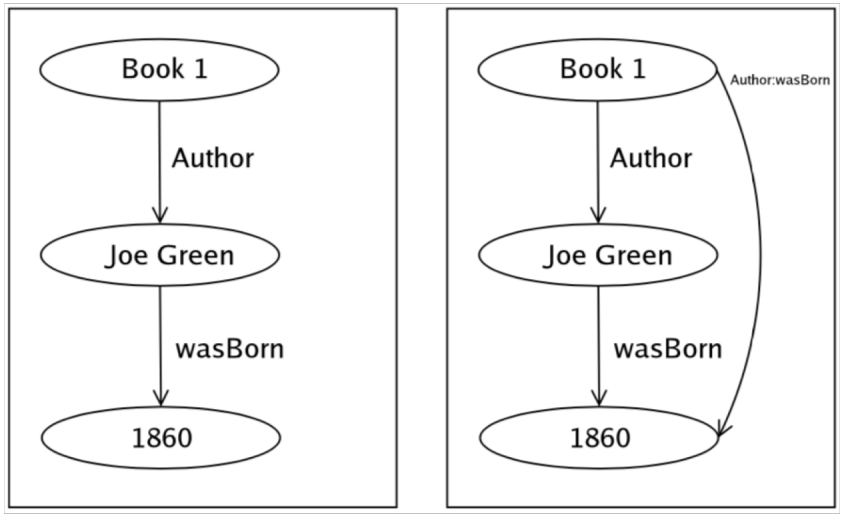
- Overcomes bandwidth waste that happens in row store DBMS as projection occurs for free – only those columns relevant to a query need to be read.
- Column-oriented data compression
- Tuple headers are stored separately

3) Materialized Views

Methodology: Pre-calculating path expression so as to avoid performing a join at query time.

Disadvantages:

- Expensive to have so many materialized views.
- Slow inserts.



Comparison of performance of 7 types of queries:

1) How many unique "property" types are there?
2) For a selected "property", (*Text*) how many other Unique "property" exist?
3) For the selected "property", which other objects are popular?
4) After selecting one more "property" (*Language*), what are the new counts?
5) How is the performance on a query that required join? (Type *Records* that originated from *US Library of Congress*)

+ 2 other complex queries

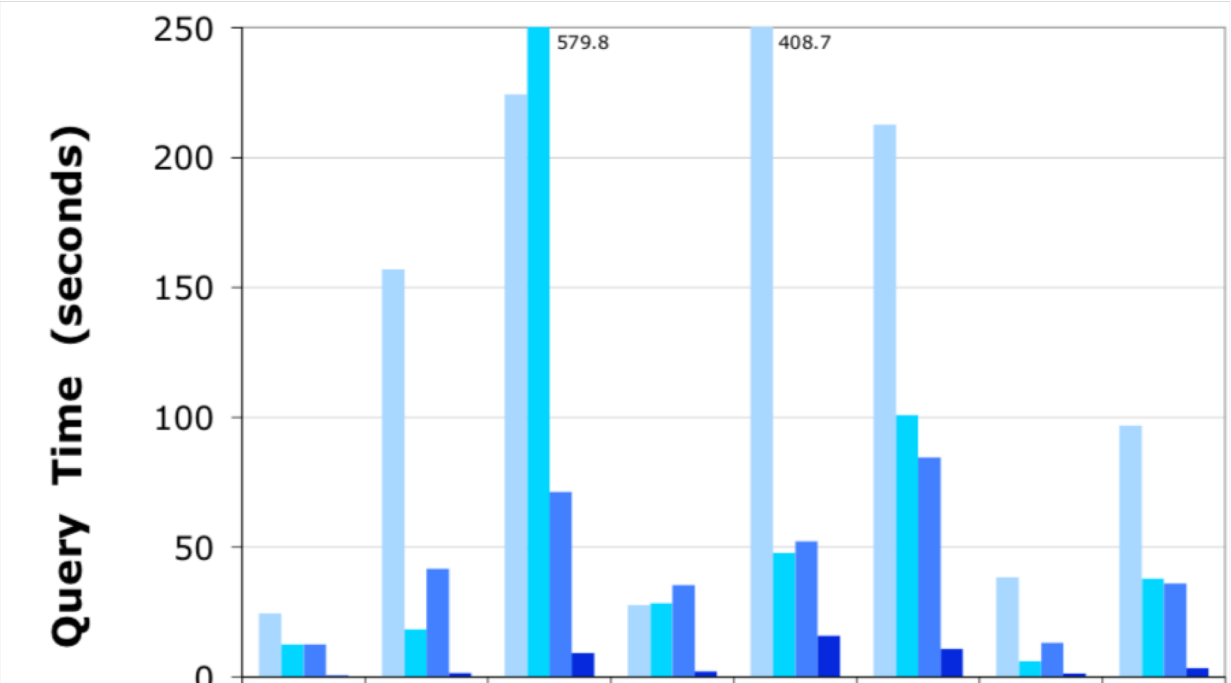
Databases Used:

Sesame – used as Triple Store

PostgreSQL – used for implementing Property Table (for both row based and column based)

C-Store – used as Column Oriented Store

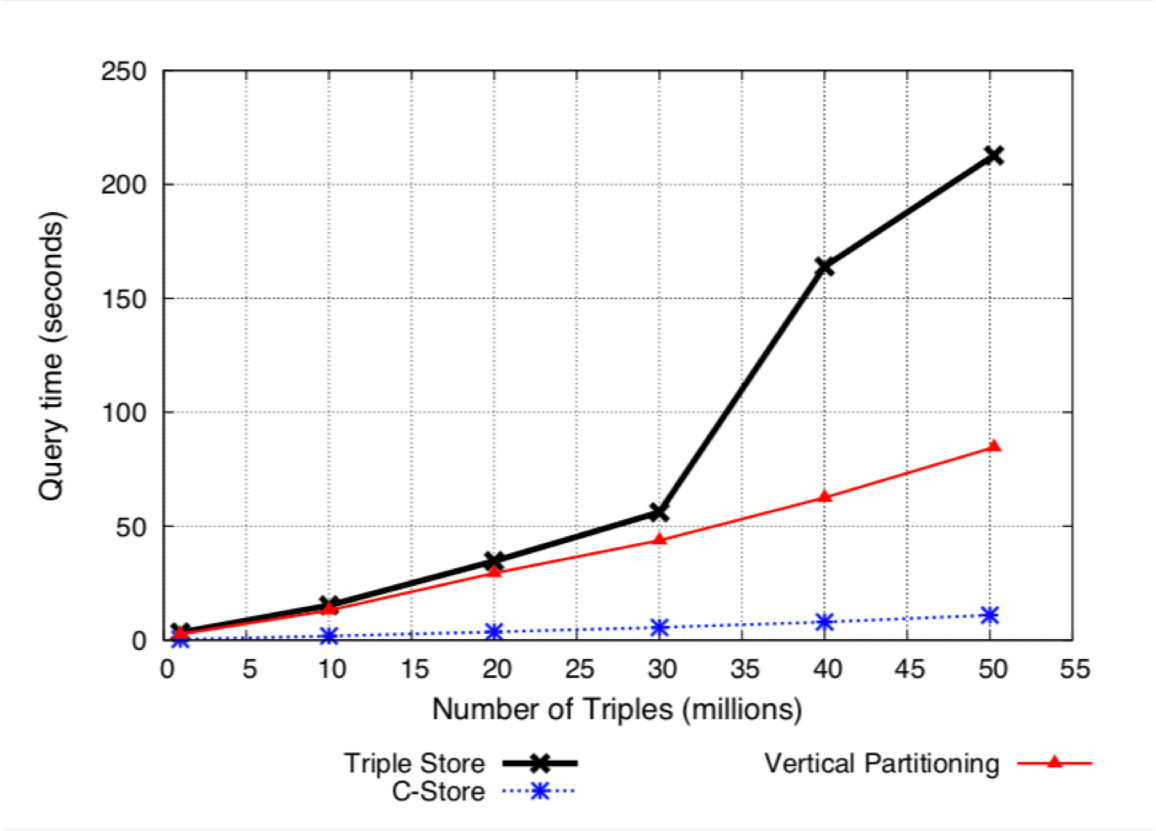
Result:



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Geo. Mean
Triple Store	24.63	157	224.3	27.67	408.7	212.7	38.37	97
Prop. Table	12.66	18.37	579.8	28.54	47.85	101	6.1	38
Vert. Part.	12.66	41.7	71.3	35.49	52.34	84.6	13.25	36
C-Store	0.66	1.64	9.28	2.24	15.88	10.81	1.44	3

Queries run even faster if paths are materialized.

Scaling:



Triple Store: Super-Linearly
Postgres and C-Store: Linearly

USC Viterbi
School of Engineering

THANK YOU!